EXHIBIT 34

THE MASC COMPOSABLE COMPUTING INFRASTRUCTURE FOR INTELLIGENT ENVIRONMENTS

Sandeep Chatterjee and Srinivas Devadas
{sandeep, devadas}@lcs.mit.edu

MIT Laboratory for Computer Science
545 Technology Square, NE43-212
Cambridge, MA, 02139, USA

Abstract. *We present a system architecture and framework for creating rapidly deployable intelligent environments. The rapid pace of innovation of computer hardware and intelligent systems software leads to uncertainty that deters manufacturers from adopting a single processor, network, or software environment for placement into their products. The MASC Composable Computing infrastructure addresses these issues by providing an upgradable hardware and software infrastructure that supports rapid development and deployment, as well as simple and economical maintenance of intelligent environment systems.*

I. Introduction

There is presently tremendous interest in building intelligent and interactive environments. The availability of cheap computing systems, networks, and appliances (comprising sensors and actuators) is enabling the development and deployment of these environments [11]. Such environments offer the promise of moving computing from a simple productivity tool to a pervasive value-added system that is a part of our daily lives [1, 8-10, 13, 15].

The setting of an intelligent environment—whether within a home, throughout an office complex, or in an industrial plant—affects the environment's systems-level architecture. For example, neither embodied robots nor video camera (observation) based intelligent environments, such as that described in [7], would be readily accepted into typical homes. Intelligent environments within homes must be seamless and non-intrusive. One approach is based on the framework of adding computational intelligence to common household information appliances, such as televisions, stereos, and telephones. Such a system relies on the hypothesis that information appliances provide a set of essential points from which to "observe" and readily interact with members of a household [3, 5, 14].

Many office environments also have societal norms and ethical implications that dictate the architecture and individual components of the intelligent environment system.

Intelligent environments for manufacturing, on the other hand, must be optimized for efficiency, safety, speed, budgetary concerns, and the need to run continuously (24-hours a day, 365 days a year).

Whatever the setting of the intelligent environment, the system may be centralized or it may be distributed. The primary determinant may be the resource demands of the applications of the environment. For example, to lessen network bandwidth utilization, vision analysis systems may be based on local computation to decode and analyze video streams. Conversely, lower bandwidth devices (sensors and actuators) may be coupled together over a network and connected to a single set of computing resources.

Intelligent environments for various settings will have some differing requirements, while sharing some overlapping qualities. However, current state-of-the-art intelligent systems are, for the most part, very specialized, closed-loop feedback systems that are expensive and complex to install, and are optimized for the environment dynamics at installation time. Since the dynamics of all environments vary and drift over time, these installed systems frequently become useless and need to be reconfigured at great expense every few years.

An intelligent environment system that is comprised of connecting together various hardware and software building blocks would be attractive. Such a *composable* system would leverage the similarities of all intelligent environments, while providing user-configurable "hooks" to tailor the environment to the particular needs of the environment or to those of its inhabitants.

To this end, a set of generalized and composable intelligence mechanisms must be developed. These

include a monitoring mechanism to sample inputs or internal state, a feedback mechanism to enable control adaptation or evolution, a caching mechanism to utilize past knowledge, and a pattern recognition mechanism to relate input/output data streams to control parameters. These mechanisms are basic to intelligent control, yet time after time considerable effort goes into implementing such mechanisms into each application-specific control task.

Not only must the software infrastructure be composable, but also the hardware infrastructure. A composable or "building-block" approach to hardware will facilitate supporting varying software configurations and their hardware resource requirements. Providing these composable software and hardware mechanisms within the environment allows for easy control specification, as well as enabling software, hardware, or hybrid implementation so as to achieve optimum cost or performance.

In this paper, we discuss the development and deployment of such a composable intelligent environment. In particular, we discuss the development of a composable software infrastructure for control automation, which enables the synthesis of efficient, adaptive control strategies across multiple different application domains. We also describe the development of a composable hardware infrastructure that allows users to simply connect various computing resource building blocks together to achieve a desired systems architecture.

II. Cross-Domain Intelligence Mechanisms

Adaptive control systems have been extensively used in applications such as automobile engine control, handwriting recognition, and speech processing for many years. However, despite isolated successes, significant effort is required to construct an adaptive control strategy for each new application.

Moreover, it is usually the case that the control strategy is tailored for a particular context. For example, different ranges of ambient light correspond to different contexts for the application of autonomous vehicle navigation. A change in contexts will frequently require significant re-engineering work, since efficient, automated control mandates the utilization of context-specific information. A control system for an autonomous vehicle that works well during daytime will require significant modification to work at night.

It is possible to learn the new context automatically. The field of Artificial Intelligence (AI) has produced general-purpose automated control methodologies such as machine learning, neural networks, and probabilistic reasoning which have been applied to a variety of control problems. Parameters of a chosen control framework are tuned upon encountering new contexts, utilizing feedback about the correctness of past decisions, in an effort to evolve the controller for improved performance. However, parameter-based control frameworks do not incorporate mathematical models of system behavior, which are necessary for efficient automation in many applications.

We propose the development of an intelligent environment for control automation, which enables the synthesis of efficient, adaptive control strategies across different application domains. We call these Cross-Domain Intelligence Mechanisms (CDIMs).

This environment consists of:

1. Mechanisms that embody application- and context independent intelligence, and
2. A language for coding control algorithms, which utilize the built-in mechanisms.

The mechanisms correspond to a three-phase control synthesis approach of Watch, Reason, and Automate. In particular, a monitoring mechanism to sample inputs or internal state, a feedback mechanism to enable control adaptation or evolution, a caching mechanism to utilize past knowledge, and a correlation mechanism to relate input/output data streams to control parameters are required. These mechanisms are basic to intelligent control, yet time after time considerable effort goes into implementing such mechanisms into each application-specific control task. The compactness of such a specification stems from the utilization of built-in constructs and mechanisms for implementing basic control functions. The efficiency of the control algorithm specified by the language is also a product of the constructs and mechanisms.

Consider an intelligent home application. The large amount of input data is filtered using simple language constructs, improving control algorithm efficiency. It is assumed that library functions for hashing are available in the language to compactly specify the caching of input values, output values, and control parameters. The hashing may be implemented in software, or as a hardware cache, but this will be transparent to the programmer.

The feedback mechanism of updating the cache entries based on actual cost data is also specified in the language. This involves tagging the control decisions and associated predicted costs, and propagating these tags through to the actual costs. Propagation is necessary to relate the actual cost to the appropriate cache entry. Tagging of decisions and costs can be automatically performed using a built-in mechanism.

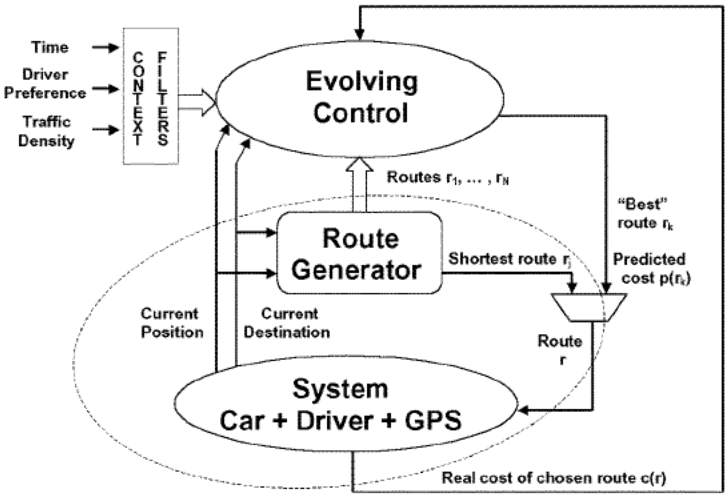

Figure 1: Intelligent Navigation

The evolution of the control strategy may follow many different trajectories. A particular trajectory can be compactly specified in the language, for example, by cost prediction functions, time series forecasting, curve fitting such as least squares, and correlation functions. Internal state in the control module may be monitored, and also serve as a basis for control decisions. Complex control based on mathematical models of system behavior can be coded in the language in the same manner as in a conventional programming language.

III. Typical Deployment Scenarios

In this section, we give detailed examples of systems that may be built with our CDIM composable intelligence system.

A. Intelligent Navigation

We consider a problem of navigation in the context of automobiles. One or more persons are driving an automobile equipped with GPS (Global Positioning Satellite) on a daily basis. GPS technology knows the position of the car at any time instant, and given destination coordinates, is able to compute a route, or routes, based on simple metrics such as shortest distance to destination. Current state-of-the-art corresponds to an open-loop system with no feedback or memories, where factors such as driver preferences, traffic conditions, and time-of-day, are ignored in the selection of the route.

Formal Statement of Problem

We consider a navigation system to be comprised of a manned vehicle, a GPS system, and a control module. GPS tracks the current position p of the vehicle at each time instant. The current destination d is an input to the system from the driver. A route r_i corresponds to a path from p to d that does not violate traffic rules. The cost of a route r_i, call it $c(r_i)$, can be computed in different ways. The simplest cost metric corresponds to the distance traversed when taking route r_i ; this is a static quantity. However, a more pertinent cost metric is the time it takes to traverse r_i, which depends on dynamically changing traffic conditions.

Strategy

An intelligent navigation system should choose a route based on many factors, including driver preference and current traffic conditions. There are two basic ways in which route computation can be intelligent, using past knowledge, or by sampling additional pertinent input. The former requires the system to have memory and feedback, and the latter may require instrumentation beyond vehicle positioning. Driver preference corresponds to an example of the utilizing past knowledge, and utilizing information about current traffic conditions corresponds to an example of additional input.

Our basic strategy for intelligent navigation is summarized in Figure 1. We assume that a "black box" route generator exists which can compute multiple routes r_1, …, r_N which represent ways of reaching a destination position from a given current position. (From a theoretical point of view, route computation is equivalent to graph traversal.) The dotted oval

represents our view of existing GPS navigation systems, where the route generator produces a single route based on the current position and current destination.

Intelligent navigation may sample additional inputs such as time of day, day of the week, traffic conditions, and current driver preference. These inputs may have to be filtered prior to being useful in making control decisions. For example, the particular day may not be relevant, what may be relevant is whether the day is a weekday or not. Traffic densities on streets may not be relevant, the useful densities typically correspond to highway entry ramps or exits.

The route generator will generate multiple routes r_1, …, r_N which will be evaluated using different criteria by the control module. Evaluation criteria will depend on distance as well as the additional input(s) described above. An important evaluation criterion of past knowledge is enabled by the feedback loop from the car and driver to the control module, which tells the control module the actual route cost, i.e., the real time taken to traverse the route, once the route is completed. Obviously, this can only be done for routes that are chosen.

It is this feedback loop that allows the control to evolve, and become more intelligent with repeated use. The control module chooses the "best" route from amongst r_1, …, r_N by predicting the cost of each route $c(r_i)$. If route r_k is chosen and traversed, then the actual time required to traverse r_k is fed back to the control module. The control module then compares the time component of the predicted cost $c(r_k)$ to the actual time, and the prediction function is corrected so as to produce better estimates upon encountering (parts of) route r_k again.

Each route r_i consists of a set of legs l_{i1}, …, l_{iM}. The cost $c(r_i)$ of a route r_i is assumed to be cumulative, i.e., $c(r_i)$ is the sum of the $c(l_{ij})$. Since the time taken to traverse a set of legs and the distance traversed in a set of legs are both cumulative, this assumption is tenable. We will assume in the sequel that the cost reflects the time taken traversing a leg, or a route.

A lookup table or a cache is implemented in the control module. The cache is initially empty. The control module produces a predicted cost or time for a route r_i namely, $p(r_i)$ by summing the $p(l_{ij})$. This prediction function p is based on the filtered inputs, and stored maps that provide information about each leg, l_{ij}, including distance. Each entry in the cache is indexed by a tuple $< l_{ij}, input >$, where $input$ corresponds to the current input condition. Each entry stores $p(l_{ij})$ or $c(l_{ij})$ which corresponds to the actual time required to traverse l_{ij}. Note that an entry is created when a cost estimate for a leg needs to be returned, and is filled initially with the predicted cost. It is updated with the real cost when the leg is actually traversed. The algorithm to utilize past knowledge simply uses the cache prior to returning predictions.

Adaptive control is achieved by evolving the prediction function. Actual costs of traversing legs are again the primary source of information to improve the prediction function. Characteristics of inputs that result in mispredictions are recognized and taken into account. Correlation between two or more legs can be discovered via static analysis, and a correlator can change the predicted cost of a leg, given the actual cost of another leg, which is deemed to be strongly correlated to the first leg. A correlator can be run periodically on the cache entries to reflect recently determined actual costs.

A possible change in context could correspond to a different driver with different preferences. For example, a particular driver may prefer to drive on highways rather than streets. A context switch can either be a human input to the system, or automatically learnt. The former is more efficient since cost predictions can be stored for different contexts or appropriately computed, whereas in the latter case predictions are initially inaccurate and have to be refined using the feedback loop.

B. Intelligent Manufacturing

Problems in manufacturing fall into different classes. In *batch manufacturing*, raw materials are provided at the start of the job, and the final product is created at the end of the job. Examples of batch manufacturing are pulp processing, and food manufacturing. *Continuous manufacturing*, for example, cement or steel manufacture, is a 24-hour, 365-days-a-year process. Here, raw materials are constantly added, and the final product is constantly created.

In batch and continuous manufacturing, the goal of intelligent manufacturing is to maximize some objective function composed from manufacturing cost, time, and product quality.

Formal Statement of Problem

We consider a continuous manufacturing system to be comprised of a plant, a control module, and a plant operator. The system continuously samples sensors x_j and displays them to an operator; different sensors may be sampled at different rates, but in general the x_j's can be viewed as samples of a time series, $x_i(t)$, where t is time.

The output of the plant, $Z(t)$, is a complicated objective function, that needs to be optimized. Typically, the value of $Z(t)$ is only known after the fact. For example, in cement manufacture, the quality of the

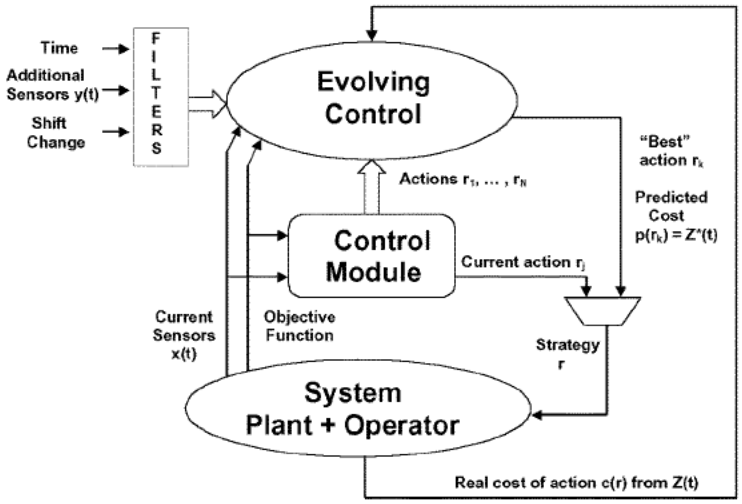

Figure 2: Intelligent Manufacturing

produced cement in only known 12-24 hours later, after samples are tested at a laboratory. Even if the output can be directly measured once the product is produced, the value of $Z(t)$ is typically influenced by actions that took place earlier in the manufacturing process. This effectively means that $Z(t)$ cannot be used in a feedback loop to control the process, because the value of $Z(t)$ may be unrelated to the current state of the process.

Strategy

In this discussion we focus on two simple aspects of intelligent manufacturing.

§ An intelligent manufacturing system should pick control actions more accurately based on experience.

§ An intelligent manufacturing system must be able to deal with sensor failure.

Our basic strategy is summarized in Figure 2. We assume that a "black box" exists which generates control actions, r_i. (The r_i's could have continuously varying parameters, in which case they should be viewed as a parameterized set of actions; for the purpose of this discussion, assume that there are a finite number of control actions). The control module selects the action, r_k, deemed to be the most useful, based on a predictive cost estimate, $p(r_i)$.

The actual cost $c(r_i)$, can be determined after the fact, once the real $Z(t)$ has been measured.

The intelligent manufacturing system can improve its ability to choose control actions by adjusting its predictive cost function based on actual measurements of $Z(t)$. It can also learn an accurate estimator $Z^*(t)$, which would be a function of values that are measured, such as the $x_i(t)$'s, other sensors $y_i(t)$ (see below), and additional variables that may be measured infrequently. Note that the evolving control will deal not only with short-term context changes, but also long-term context drift, and the latter will be indicated by lowered output $Z(t)$.

Typically, manufacturing plants are over-instrumented with sensors. The control module typically uses a much smaller fraction of these sensors (the x_j's) to make its decisions. However, additional sensors, y_j's, can be used to compensate for sensor failure. For example, if a sensor x_1 fails, from historical data, we can learn how to estimate x_1 from the values of sensors that have not failed (e.g., other x_j's or the y_j's). In fact, information about sensor failure becomes part of the context, and the intelligent manufacturing system either switches to a different set of control strategies designed to work in the presence of missing information, or requests additional information to estimate the missing sensors.

The very nature of composable or configurable software demands a similarly composable hardware platform. As the hardware resource requirements of the software environment change (e.g., because of configuring the software differently), hardware mechanisms must be provided to enable the user to easily and economically upgrade or replace the underlying computing and peripheral hardware resources. We describe such a mechanism next.

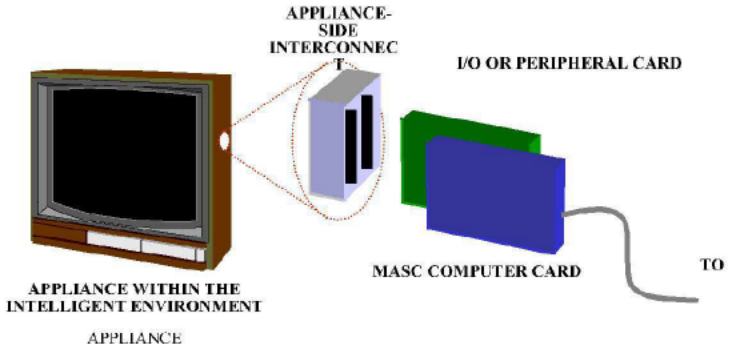

Figure 3: MASC Composable Computing Hardware Platform Based Intelligent Environment.

IV. The MASC Composable Computing Hardware Platform

The MASC Composable Computing hardware platform addresses the need for a modular computation device for intelligent systems and environments [2, 4, 6].

As shown in Figure 3, MASC is comprised of the five following components:

- MASC computer cards
- Appliances with MASC appliance-side connectors
- Software running atop MASC cards
- Internal network
- Gateway to external networks

MASC cards form the computation and communications hardware platform of our approach. Each card synthesizes the properties of "smart cards" and PCMCIA [12] cards (PC cards), and contains within it a microprocessor, a network adapter, main memory, and non-volatile storage, and implements a common interface with which to communicate with all host appliances[1].

Microprocessor. Any standard microprocessor can be used as the computation engine for MASC. Furthermore, different implementations may emphasize different performance characteristics, e.g., speed versus power consumption, to suit different applications.

Network Adapter. Any network can be used as the vehicle for communication between different MASC cards. Accordingly, different implementations may represent different combinations of processor and internal network architecture. MASC's network adapter can be used to provide connectivity between different appliances, and to download new software onto the card.

Memory and Non-volatile Storage. Each MASC card contains memory for use by the processor and some local non-volatile storage. Again, the type and amount of memory and storage can be tailored to meet the demands of different applications. The non-volatile storage allows application programs and user data to be saved locally. Additional application programs can be downloaded over the network and executed in a manner similar to that used in a network computer.

Interface Block. The interface block allows communication between the MASC card and the appliance-side connector. When a MASC card is connected to an appliance, the interface block deciphers the type of the appliance and how to communicate with it. After this, its primary task is to send data and commands between the MASC card and the appliance socket.

As newer computation engines and networks become available, the *user* can simply remove the MASC card from the appliance and replace it with a new one. Essentially, the architecture of the MASC allows three *degrees of freedom* (in choosing and upgrading 1. the processor, 2. the network and 3. the software environment) which can be exploited to take advantage of the areas of greatest technological change and improvement. MASC's modular and user-swappable cards are essential because software upgrades often first require hardware upgrades. As software becomes increasingly complex, the most common hardware components for upgrading are processors and memory (including L2 cache).

V. Conclusions

This paper presented a novel framework and paradigm for creating *rapidly deployable* intelligent environments for various settings, including homes,

[1] The network adapter and non-volatile storage may be placed onto separate cards in order to enhance the modularity and flexibility of the approach.

offices, and industrial manufacturing plants. We argued that by using composable building blocks for the hardware and software architecture, benefits including easier development, simpler installations and maintenance, and lower costs might be achieved. To this end, we described our composable computing hardware and software infrastructure. Furthermore, we showed that applications of significant import can be created with our framework.

VI. Acknowledgements

We thank R. Bharat Rao of Siemens Corporate research for discussions regarding intelligent manufacturing. This work was supported in part by a grant from NTT, Atsugi, Japan.

References

1. Abowd, G., *et al. Teach and learning a Multimedia Authoring: The Classroom 2000 Project*. in Proceedings of *ACM Multimedia'96 Conference*. 1996.
2. Chatterjee, S. *The MASC Network Architecture: A Novel Paradigm of Computing Through Information Appliances*. in Proceedings of *the First IEEE International Symposium on Consumer Electronics*. 1997. Singapore.
3. Chatterjee, S. *SANI: A Seamless and Non-Intrusive Framework and Agent for Creating Intelligent Interactive Homes*. in Proceedings of *the Second ACM International Conference on Autonomous Agents*. 1998. Minneapolis/St. Paul.
4. Chatterjee, S. *Towards a MASC Appliances-Based Educational Computing Paradigm*. in Proceedings of *the Thirteenth ACM Symposium on Applied Computing*. 1998. Atlanta, Georgia.
5. Chatterjee, S. *Towards Rapidly Deployable Intelligent Environments*. in Proceedings of *the Eleventh AAAI Spring Symposium on Intelligent Environments*. 1998. Stanford, California: AAAI Press.
6. Chatterjee, S. and S. Devadas, *MASC: A User-Embeddable Hardware Platform and Infrastructure for Information Appliances*, . 1999, MIT Laboratory for Computer Science TR-591: Cambridge, MA.
7. Coen, M. *Building Brains for Rooms: Designing Distributed Software Agents*. in Proceedings of *the Innovative Applications of Artificial Intelligence Conference*. 1997. Providence, RI.
8. Druin, A. and K. Perlin. *Immersive Environments: A Physical Approach to the Computer Interface*. in Proceedings of *Conference on Human Factors in Computer Systems (CHI'94)*. 1994.
9. Lucente, M., G. Zwart, and A. George. *Visualization Space: A Testbed for Deviceless Multimodal User Interface*. in Proceedings of *AAAI 1998 Spring Symposium on Intelligent Environments, AAAI TR SS-98-02*. 1998.
10. Mozer, M. *The Neural Network House: An Environment that Adapts to its Inhabitants*. in Proceedings of *AAAI 1998 Spring Symposium on Intelligent Environments, AAAI TR SS-98-02*. 1998.
11. Saffo, P., *Sensors: The Next Wave of Innovation*, in *Communications of the ACM*. 1997. p. 92-97.
12. Shanley, T., *PCMCIA System Architecture*. 2nd ed. PC System Architecture: MindShare. 464.
13. Want, R., *et al.*, *The ParcTab Ubiquitous Computing Experiment*, , Xerox Parc Technical Report.
14. Weiser, M., *The Computer for the Twenty-First Century*, in *Scientific American*. 1991. p. 94-100.
15. Weiser, M., *The World Is Not a Desktop*, in *Interactions*. 1994. p. 7-8.